Dérivation d'algorithmes sans verrou à partir d'une spécification atomique

Loïc Fejoz[1] et Stephan Merz

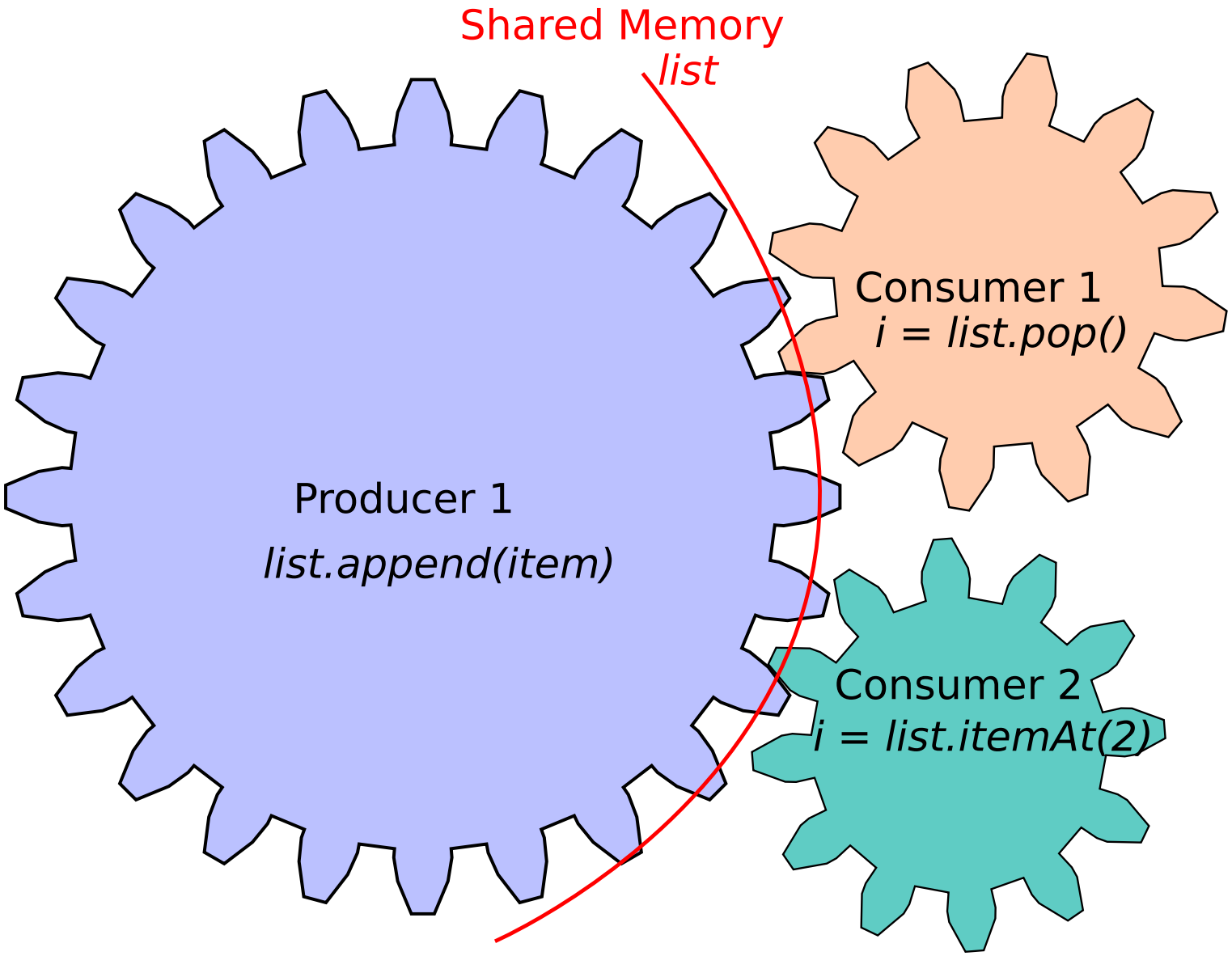
INRIA Lorraine & LORIA, Nancy, France
Loic.Fejoz@loria.fr Stephan.Merz@loria.fr

20 juin 2007

[1]This work was supported by Microsoft Research through its European PhD Scholarship Programme.

1. Contexte des algorithmes sans verrou
2. Exemples
   - Affectation simple
   - Descripteurs
3. Méthode proposée
   - Algorithme
   - Spécification
   - Raffinement
4. Conclusion & Travaux futurs

Modélisation de processus caractérisés par des algorithmes



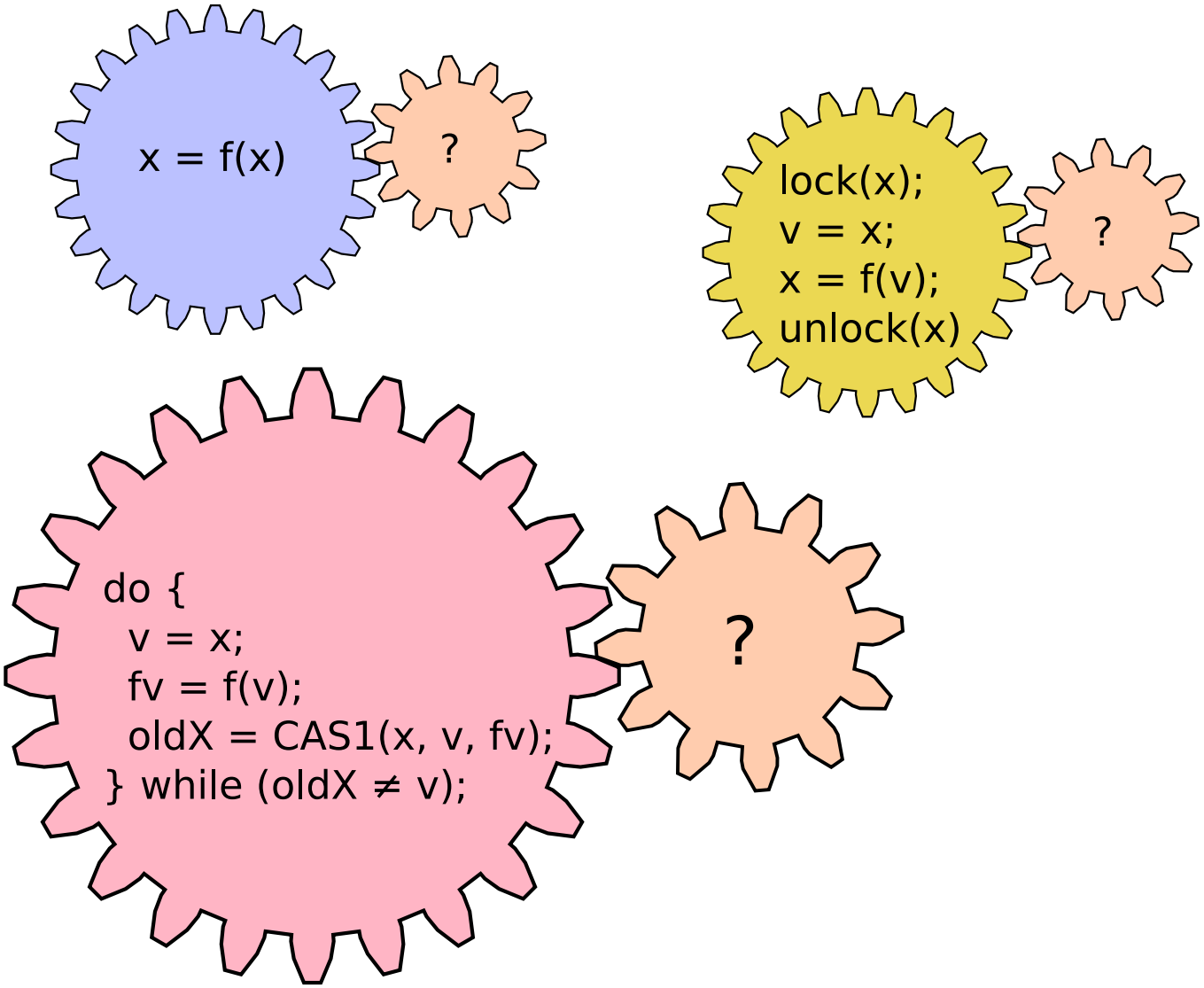
Pourquoi des algorithmes sans verrou ?

- efficacité ? souvent. Cela dépend de la contention.
- vivacité ? selon les algorithmes.
- sûreté ? oui.
- primitives atomiques enfin répandues (CAS, Test-and-Set, Load-Link/Store-Conditionnal, . . .).

Compare-and-Swap CAS

```
word_t CAS1(word_t *a, word_t o, word_t n) {
  word_t old = *a;
  if (old == o) {
    *a = n;
  }
  return old;
}
```

Affectation



Affectation avec descripteur

Affectation atomique

```
*x = *x + *a ;
```

Affectation avec descripteur

```
do {
  d = new Descriptor(a, *x) ;
  v = CAS1(x, d->x, d) ;
  if IsDescriptor(v) {
    CAS1(x, v, v->x + *(v->a)) ;
  }
} while (v ≠ d->x) ;
CAS1(x, d, d->x + *(d->a))
```

Affectation avec descripteur

Affectation atomique

```
*x = *x + *a ;
```

Affectation avec descripteur

```
do {
  d = new Descriptor(a, *x) ;
  v = CAS1(x, d->x, d) ;
  if IsDescriptor(v) {
    CAS1(x, v, v->x + *(v->a)) ;
  }
} while (v ≠ d->x) ;
CAS1(x, d, d->x + *(d->a))
```

Particularités des algorithmes sans verrou

- opérations atomiques particulières (CAS, etc),
- exécutions parallèles mais pas simultanées (*interleaving semantics*),
- composition d'algorithmes,
- descripteurs, ainsi que d'autres solutions-types.

Les méthodes

Rely-Guarantee

composition

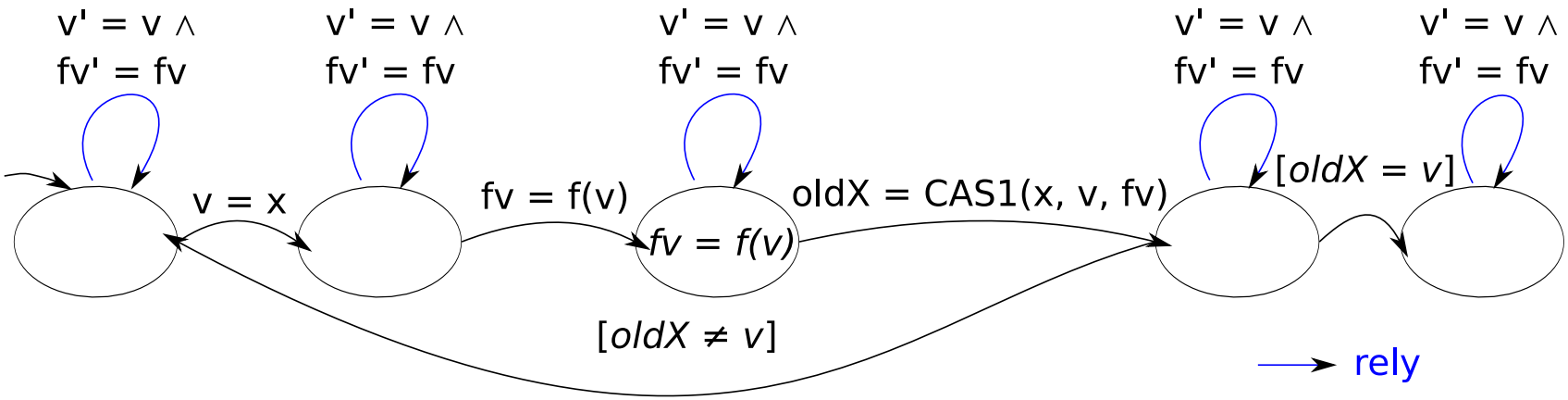
méthode B

raffinement, preuves semi-automatiques

TLA$^+$ & $^+$CAL

linearisabilité

Exemple




```
do {
  v = x;
  fv = f(v);
  oldX = CAS1(x, v, fv);
} while(oldX != v);
```

Algorithme

- Un algorithme $A = (CF, local_invariant, rely, guard, effect)$ est décrit par son contrôle de flot, les invariants locaux, une condition de garantie des invariants, une guarde, et les transitions.
- Un algorithme est donc valide ssi :
  $local_invariant \wedge guard \wedge effect \Rightarrow local_invariant'$,
- $local_invariant \wedge rely \Rightarrow local_invariant'$.

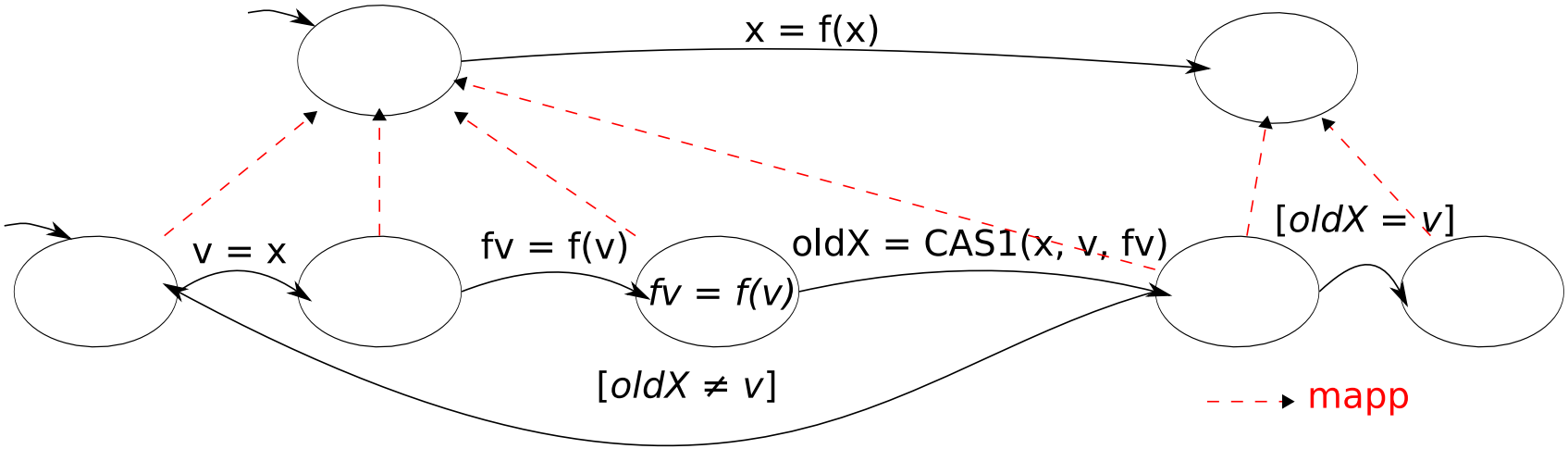
Spécification

- Une spécification est une liste d'algorithmes $Spec = [A_1, \ldots, A_n]$.
- Une spécification est valide ssi tous les algorithmes sont valides, et si chacune des actions implique les rely des autres algorithmes.

Le raffinement comme preuve de correction

- La spécification abstraite décrit l'opération atomique.
- L'implémentation est dérivée pas à pas, simplifiant les obligations de preuves.
- Les dérivations peuvent être l'application de solutions-types "pattern".

Exemple



Raffinement de spécification...

Un raffinement entre les spécifications $\underline{Spec} = [\underline{A}_1, \ldots, \underline{A}_n]$ et $\overline{Spec} = [\overline{A}_1, \ldots, \overline{A}_n]$ sous *mapp* et *gluing_invariant* est valide ssi :

- $\underline{Spec}$ et $\overline{Spec}$ sont valides.
- Tout algorithme concret $\underline{A}_i$ raffine l'algorithme abstrait $\overline{A}_i$
- Un algorithme concret $\underline{A}_j$ peut effectuer un pas abstrait d'un autre algorithme $\overline{A}_i$.

Conclusion & Travaux futurs

Conclusion

- méthode adaptée à la vérification d'une famille d'algorithmes.
- ... plus particulièrement aux algorithmes sans verrou
- ... exécutés par un nombre indéterminé de processus
- ... à mémoire partagée.
- La méthode et sa correction sont en cours de justification dans l'assistant à la preuve Isabelle/HOL.

Travaux futurs

- Application au RDCSS
- création d'une syntaxe concrète pour l'aide au développement
- création d'un générateur d'obligation de preuves s'interfaçant avec des prouveurs automatiques
- justification de l'application de solutions-types

Questions

loic.fejoz@loria.fr

RDCSS...

```
word_t RDCSS(word_t *a1, word_t o1, word_t *a2, word_t o2, word_t n2) {
  r = *a2;
  if ((r == o2) && (*a1 == o1)) {
    *a2 = n2;
  }
  return r;
}

word_t RDCSSRead(word_t *a2) {
  return *a2;
}
```

RDCSS. . . et son implémentation

```
word_t RDCSS(RDCSSDescriptor_t *d) {
  do {
    r = CAS1(d->a2, d->o2, d); // C1
    if (IsDescriptor(r)) Complete(r); // H1
  } while (IsDescriptor(r)); // B1
  if (r == d->o2) Complete(d);
  return r;
}

word_t RDCSSRead(addr_t *addr) {
  do {
    r = *addr; // R1
    if (IsDescriptor(r)) Complete(r); //H2
  } while (IsDescriptor(r)); // B2
  return r;
}

void Complete(RDCSSDescriptor_t *d) {
  v = *(d->a1); // R2
  if (v == d->o1) {
```